```
Note - The path specified in RDD and Data frame transformations is relative. Use absolute path to perform the same without fail.

## Exercise 1 (Importing Data - Exercise)  Solution
DimEmployee_RDD = sc.textFile("./dataset/AdventureWorks_RDD/DimEmployee.csv")
DimProduct_RDD = sc.textFile("./dataset/AdventureWorks_RDD/DimProduct.csv")
FactResellerSales_RDD = sc.textFile("./dataset/AdventureWorks_RDD/FactResellerSales.csv")

## Exercise 2 (Split Data Based on Delimiter - Exercise) Solution
DimEmployee_RDD = DimEmployee_RDD.map(lambda var1 : var1.split(","))
DimProduct_RDD = DimProduct_RDD.map(lambda var1 : var1.split(","))
FactResellerSales_RDD = FactResellerSales_RDD.map(lambda var1 : var1.split(","))

## Exercise 3 (Creating Key-Value Pairs in RDD - Exercise) Solution
blank = ['','NA'] #required for replacing missing values like '' and 'NA'
DimEmployee_RDD= DimEmployee_RDD.map(lambda x:[u'0' if i in blank else i for i in x])
DimEmployee_RDD_split = DimEmployee_RDD.map(lambda x:(float(x[0]),list([float(x[1]),float(x[2]),x[3]])))

DimProduct_RDD= DimProduct_RDD.map(lambda x:[u'0' if i in blank else i for i in x])
DimProduct_RDD_split = DimProduct_RDD.map(lambda x:(float(x[0]),list([float(x[1]),x[2],float(x[3]),x[4],float(x[5]),float(x[6])])))

FactResellerSales_RDD= FactResellerSales_RDD.map(lambda x:[u'0' if i in blank else i for i in x])
FactResellerSales_RDD_split = FactResellerSales_RDD.map(lambda x:(float(x[0]),list([float(i) for i in x[1:]])))

## Exercise 4 (Sorting RDD Data Based on Key - Exercise) Solution
DimEmployee_RDD_split_sort = DimEmployee_RDD_split.sortByKey(ascending=True,keyfunc=lambda k: k)
DimProduct_RDD_split_sort = DimProduct_RDD_split.sortByKey(ascending=True,keyfunc=lambda k: k)
FactResellerSales_RDD_split_sort = FactResellerSales_RDD_split.sortByKey(ascending=True,keyfunc=lambda k: k)

## Exercise 5 (RDD Joins - Exercise) Solution
reseller_prod = DimProduct_RDD_split_sort.join(FactResellerSales_RDD_split_sort)

## Exercise 6 (Filtering Data - Exercise) Solution
reseller_prod_filter = reseller_prod.filter(lambda x: x[1][1][13] > 3200)

## Exercise 7 (Counting - Exercise) Solution
a=sc.parallelize(reseller_prod.countByKey().items()).sortBy(lambda x: x[1],ascending=False).take(10)
a=[i[0] for i in a]
reseller_prod.filter(lambda x: x[0] in a).map(lambda x: x[1][0][1]).distinct().collect()

## Exercise 8 (Distinct - Exercise) Solution
reseller_prod_mod=reseller_prod.map(lambda x:x[0])
b=reseller_prod_mod.distinct().collect()
reseller_prod.filter(lambda x: x[0] in b).map(lambda x: x[1][0][1]).distinct().collect()

## Exercise 9 (Aggregating Grouped RDD - Exercise) Solution
reseller_prod_group=reseller_prod.map(lambda x:(x[0], x[1][1][13]))
reseller_prod_top=reseller_prod_group.groupByKey().mapValues(sum).sortBy(lambda x: x[1],ascending=False)
c=reseller_prod_top.take(5)
```

```
c=[i[0] for i in c]
reseller_prod.filter(lambda x: x[0] in c).map(lambda x: x[1][0][1]).distinct().collect()

## Exercise 10 (Calculating Minimum Maximum and Mean of Data - Exercise) Solution
d=reseller_prod_top.map(lambda x:x[1]).min()
e=c=reseller_prod_top.filter(lambda x: x[1]==d).collect()
e=[i[0] for i in e]
reseller_prod.filter(lambda x: x[0] in e).map(lambda x: x[1][0][1]).distinct().collect()

## Exercise 11 (Summary of Data - Exercise) Solution
reseller_prod_filter.map(lambda x:x[1][1][13]).stats()

#Exercise 12 (Import Data as Data Frame - Exercise) Solution
DimDate = spark.read.csv(path="./dataset/AdventureWorks_DF/DimDate.csv",sep = ',',header = True)
DimEmployee = spark.read.csv(path="./dataset/AdventureWorks_DF/DimEmployee.csv",sep = ',',header = True)
DimGeography = spark.read.csv(path="./dataset/AdventureWorks_DF/DimGeography.csv",sep = ',',header = True)
DimProduct = spark.read.csv(path="./dataset/AdventureWorks_DF/DimProduct.csv",sep = ',',header = True)
DimProductCategory = spark.read.csv(path="./dataset/AdventureWorks_DF/DimProductCategory.csv",sep = ',',header = True)
DimProductSubcategory = spark.read.csv(path="./dataset/AdventureWorks_DF/DimProductSubcategory.csv",sep = ',',header = True)
DimReseller = spark.read.csv(path="./dataset/AdventureWorks_DF/DimReseller.csv",sep = ',',header = True)
DimSalesTerritory = spark.read.csv(path="/home/ajay_trng/dataset/AdventureWorks_DF/DimSalesTerritory.csv",sep = ',',header = True)
FactResellerSales = spark.read.csv(path="./dataset/AdventureWorks_DF/FactResellerSales.csv",sep = ',',header = True)

## Exercise 13 (Convert RDD to Data Frame - Exercise) Solution
DimProduct_RDD = sc.textFile("./dataset/AdventureWorks_RDD/DimProduct.csv")
DimProduct_RDD = DimProduct_RDD.map(lambda var1 : var1.split(","))
DimProduct_DF=spark.createDataFrame(DimProduct_RDD)
DimProduct_DF.show(10)

## Exercise 14 (Joins - Exercise) Solution
join_df=DimProductCategory.join(other=DimProductSubcategory,on='ProductCategoryKey',how='inner').join(other=DimProduct,on='ProductSubc
ategoryKey',how='inner').join(other=FactResellerSales,on='ProductKey',how='inner')
product_sales=join_df.select("EnglishProductCategoryName", "EnglishProductSubcategoryName", "EnglishProductName", "SalesAmount")

##Exercise 15 (Converting Datatype - Exercise) Solution
final_df=join_df.select(join_df.ProductKey.cast("float"),join_df.ProductSubcategoryKey.cast("float"),join_df.ProductCategoryKey.cast("
float"),join_df.ProductCategoryAlternateKey.cast("float"),join_df.EnglishProductCategoryName,join_df.EnglishProductSubcategoryName,joi
n_df.EnglishProductName,join_df.StandardCost.cast("float"),join_df.Color,join_df.ListPrice.cast("float"),join_df.DealerPrice.cast("flo
at"),join_df.OrderDateKey.cast("float"),join_df.DueDateKey.cast("float"),join_df.ShipDateKey.cast("float"),join_df.ResellerKey.cast("f
loat"),join_df.EmployeeKey.cast("float"),join_df.PromotionKey.cast("float"),join_df.SalesTerritoryKey.cast("float"),join_df.OrderQuant
ity.cast("float"),join_df.UnitPrice.cast("float"),join_df.UnitPriceDiscountPct.cast("float"),join_df.DiscountAmount.cast("float"),join
_df.ProductStandardCost.cast("float"),join_df.TotalProductCost.cast("float"),join_df.SalesAmount.cast("float"),join_df.Freight.cast("f
loat"))

## Exercise 16 (Sorting Data - Exercise) Solution
from pyspark.sql.types import StructType, StructField, FloatType, StringType,IntegerType
buys_schema = pyspark.sql.types.StructType([StructField("age", IntegerType()),StructField("income", FloatType()),StructField("gender",
StringType()),StructField("marital", StringType()),StructField("buys", StringType())])
buy1 = spark.read.csv(path='./dataset/buy.csv',sep=',',header=True,schema=buys_schema)
```

```
buy1.sort('income', ascending=True).show()

## Exercise 17 (Filtering Data - Exercise) Solution
product_filter=product_sales.filter("EnglishProductCategoryName=='Accessories'")

## Exercise 18 (Count of Values - Exercise) Solution
product_filter.count()

## Exercise 19 (Aggregation - Exercise) Solution
part (a) final_df.agg({"SalesAmount": "mean"}).show()
part (b) final_df.groupBy("EnglishProductCategoryName").agg({'SalesAmount':'mean'}).show()

## Exercise 20 (Multi-Dimension View of Data - Exercise) Solution
final_df.select('SalesAmount','DiscountAmount','EnglishProductCategoryName').cube('EnglishProductCategoryName').mean().show()

## Exercise 21 (Co-variance and Correlation - Exercise)
print('co-variance - ' , final_df.cov('SalesAmount','DiscountAmount'))
print('correlation - ' , final_df.corr('SalesAmount','DiscountAmount'))

## Exercise 22 (Querying Temp Table - Exercise) Solution
final_df.createOrReplaceTempView("final_temp")
sql1 = spark.sql("SELECT EnglishProductCategoryName,EnglishProductSubcategoryName,avg(SalesAmount) from final_temp group by
EnglishProductCategoryName,EnglishProductSubcategoryName")
sql1.show()

## Exercise 23 (Accessing Hive Tables - Exercise) Solution
sqlContext.sql('create database db') #here db is database name
sqlContext.sql('use db')
from pyspark.sql import DataFrameWriter
dfw = DataFrameWriter(final_df)
dfw.saveAsTable(name="final_hive",mode='overwrite')
sqlContext.sql('select * from final_hive').show()
sqlContext.tableNames()

## Exercise 24 (Implementing UDAF - Exercise) Solution
from pyspark.sql.types import FloatType
from pyspark.sql import UDFRegistration
udf1 = UDFRegistration(sqlContext)
udf1.register(name='fun_sum', f=lambda var1: sum(var1), returnType=FloatType())
del1 = sqlContext.sql('SELECT * FROM final_hive')
from pyspark.sql import functions as f
final_hive_list = del1.agg(f.collect_list(del1['SalesAmount']).alias('agg_col'))
final_hive_list.createOrReplaceTempView("iris_agg")
sqlContext.sql('SELECT fun_sum(agg_col) as sum_sales_amount FROM iris_agg').show()

## Exercise 25 (Linear Regression - Exercise) Solution
from pyspark.mllib.regression import LabeledPoint
import numpy
FactResellerSales_RDD = sc.textFile("./dataset/AdventureWorks_RDD/FactResellerSales.csv")
```

```
FactResellerSales_RDD = FactResellerSales_RDD.map(lambda var1 : var1.split(","))
training1 = FactResellerSales_RDD.map(lambda var1: LabeledPoint(float(var1[14]),[float(var1[15])]))
training1.cache()
from pyspark.mllib.regression import LinearRegressionWithSGD
lm1 = LinearRegressionWithSGD.train(training1, iterations=10000, step=0.01, intercept=True)
print(lm1)
testing1 = sc.textFile("./dataset/AdventureWorks_RDD/FactResellerSales.csv")
testing1 = testing1.map(lambda var1: var1.split(","))
testing1_FR = testing1.map(lambda var1: [float(var1[15])])
testing1_SA = testing1.map(lambda var1: [float(var1[14])])
#cache data for faster execution
testing1_FR.cache()
testing1_SA.cache()
print(lm1.predict(testing1_FR).collect())

## Exercise 26 (Logistic Regression - Exercise) Solution
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
import numpy
Bank_Marketing_dataset_training = sc.textFile("./dataset/Bank_Marketing_dataset_training.csv")
Bank_Marketing_dataset_training = Bank_Marketing_dataset_training.map(lambda var1 : var1.split("\t"))
training1 = Bank_Marketing_dataset_training.map(lambda var1: LabeledPoint(float(var1[20]),
[float(var1[0]),float(var1[10]),float(var1[11])]))
lg1 = LogisticRegressionWithLBFGS.train(training1, iterations=10000, numClasses=2)
print(lg1)
testing1 = sc.textFile("./dataset/Bank_Marketing_dataset_testing.csv")
testing1 = testing1.map(lambda var1 : var1.split("\t"))
testing1 = testing1.map(lambda var1: [float(var1[0]),float(var1[10]),float(var1[11])])
p1 = lg1.predict(testing1)
print(p1.collect())

## Exercise 27 (Random Forest (Classification) - Exercise) Solution
import numpy
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import RandomForest
Qualitative_Bankruptcy = sc.textFile("./dataset/Qualitative_Bankruptcy.txt")
Qualitative_Bankruptcy = Qualitative_Bankruptcy.map(lambda var1: var1.split(","))
training1 = Qualitative_Bankruptcy.map(lambda var1: LabeledPoint(float(var1[6]),
[float(var1[0]),float(var1[1]),float(var1[2]),float(var1[3]),float(var1[4]),float(var1[5])]))
rfc1 = RandomForest.trainClassifier( data=training1 , numClasses=2, categoricalFeaturesInfo ={}, numTrees=5)
print(rfc1)
testing1 = sc.textFile("./dataset/Qualitative_Bankruptcy.txt")
testing1 = testing1.map(lambda var1: var1.split(","))
testing1 = testing1.map(lambda var1: [float(var1[0]),float(var1[1]),float(var1[2]),float(var1[3]),float(var1[4]),float(var1[5])])
p1 = rfc1.predict(testing1)
print(p1.collect())

## #Exercise 28 (Gradient Boosting Trees (Classification) - Exercise) Solution
import numpy
```

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import GradientBoostedTrees
bank_note_authentication = sc.textFile("./dataset/bank_note_authentication.txt")
bank_note_authentication = bank_note_authentication.map(lambda var1 : var1.split(","))
training1 = bank_note_authentication.map(lambda var1: LabeledPoint(float(var1[4]),
[float(var1[0]),float(var1[1]),float(var1[2]),float(var1[3])]))
gbtc1 = GradientBoostedTrees.trainClassifier( data=training1 , categoricalFeaturesInfo ={}, numIterations=20)
print(gbtc1)
testing_data = sc.parallelize([[u'-2.8829',u'-0.60324',u'2.9085',u'1.4657']])
testing_data = testing_data.map(lambda var1: [float(var1[0]),float(var1[1]),float(var1[2]),float(var1[3])])
p1 = gbtc1.predict(testing1)
print(p1.collect())
```